Process-oriented Inconsistency Management in Collaborative Systems Modeling

István Dávid[1,2] Joachim Denil[1,2] Hans Vangheluwe[1,2,3]

[1]University of Antwerp, Belgium
[2]Flanders Make, Belgium
[3]McGill University, Montréal, Canada
{istvan.david, joachim.denil, hans.vangheluwe}@uantwerpen.be

KEYWORDS

Concurrent engineering, model-based systems engineering, heterogeneous systems, model consistency

ABSTRACT

Engineered products have reached a complexity which requires explicit modeling and analysis of various aspects such as performance, safety, and energy-efficiency, before system realization. This allows stakeholders in various domains to collaborate on a "virtual product", using their domain-specific modeling languages and tools. This variety of modeling languages and tools, if not managed properly, can give rise to inconsistencies between stakeholder models, resulting in an incorrect product. Managing inconsistencies, therefore, is a key enabler to efficient collaborative engineering. Explicitly modeling the engineering process in which the virtual product models are manipulated allows for a detailed analysis of the root causes of inconsistencies and of the impact of applying various (in-)consistency management techniques on the process.

INTRODUCTION

The complexity of currently engineered systems has increased drastically over the last decades. To tackle this complexity, a virtual product can be designed before realizing the real product. The design of the virtual product is achieved by model-based techniques, and typically in heterogeneous collaborative settings. This means stakeholders of different domains interact with the virtual product through the models in their own views and viewpoints (Corley et al., 2016). Pertinent examples include the engineering of mechatronic and cyber-physical systems (CPS), and Internet-of-Things (IoT) systems, in which the hardware and software elements have to be modeled, simulated and verified together.

Such collaborative endeavors are, however, severely hindered by inconsistencies between the engineering artifacts such as domain-specific models, requirement specifications and other documents.

According to Herzig et al. (2014), *an inconsistency is present if two or more statements are made that are not jointly satisfiable.* Note, that this definition is not limited to models in a pure software settings, but is valid in multi-disciplinary engineering as well. Spanoudakis and Zisman (2001) also characterize the notion of an inconsistency using the joint non-satisfiability criterion, but in addition, they shed more light on the origin of inconsistencies by relating inconsistencies to *overlapping elements of different software models.*

In our view as well, inconsistencies stem from the joint non-satisfiability of statements. In our view, however, it is the ontological and linguistic *properties* (Vanherpen et al., 2016) of the system that cannot be the jointly satisfied. The link between the aforementioned overlaps and system properties has been first identified in previous work by Persson et.al. (Persson et al., 2013).

The problem of inconsistencies is exacerbated by the disparity of the domain-specific modeling languages and modeling tools involved in the flow of the engineering work. Stakeholders, quite often, do not use the same vocabulary when describing the various aspects of the system, resulting in overlaps between terminologies, and consequently, in their models. Because these inter-domain overlaps are not easy to identify, inconsistencies spanning the various involved domains are harder to detect.

Inconsistencies, if not managed properly, may lead to an incorrect product, which does not satisfy the properties required by the specifications. In extreme cases this may for example lead to safety breaches in mission critical systems.

Managing inconsistencies, therefore, is a must in every engineering process, especially in the collaborative ones. Wrongly executed inconsistency management, however, may have severe repercussions on the time it takes to produce a correct product. For example, dealing with incompatible sub-system interfaces during the integration phase may require additional iterations over costly engineering activities. Consequently, analyzing the impact of various inconsistency management techniques is desirable.

Explicit modeling of the engineering process enables combining the two facets of proper and efficient inconsis-

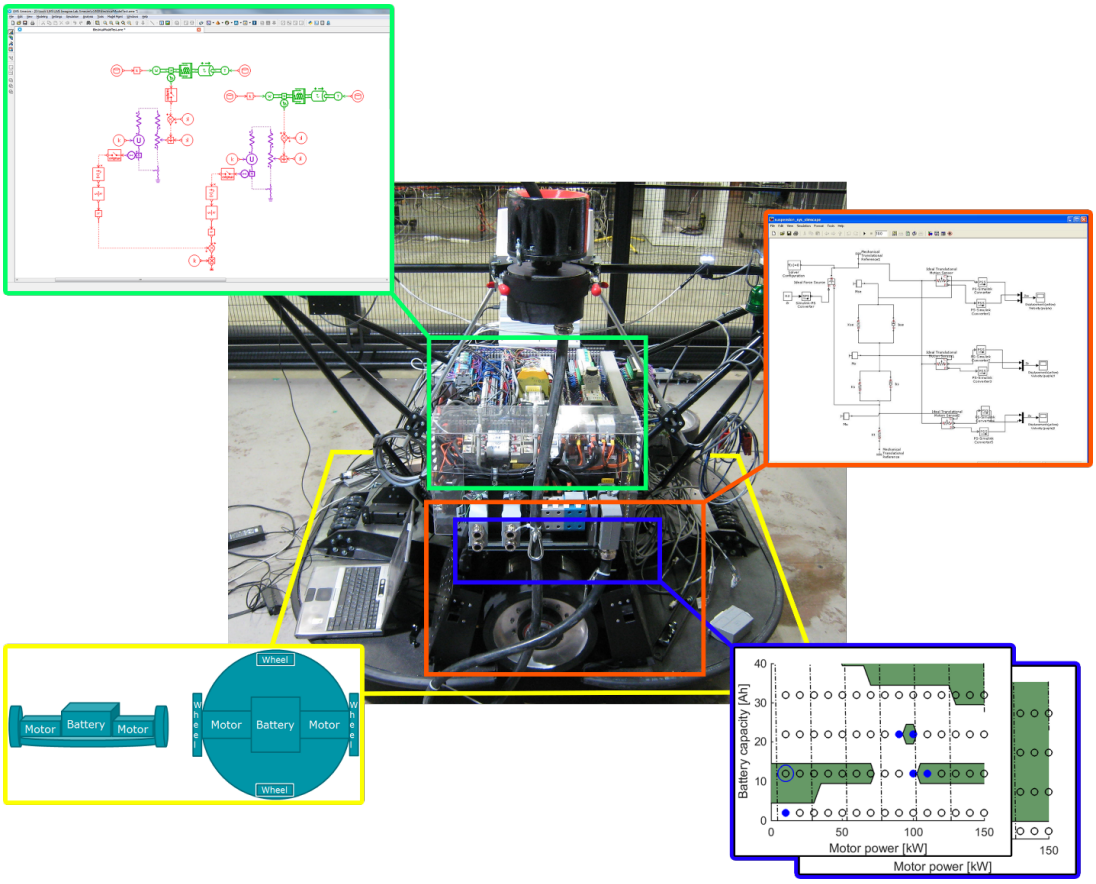

Figure 1: Autonomous Guided Vehicle (AGV): an example heterogeneous system which requires stakeholder collaboration across various domains.

tency management. On the one hand, the process model enables thorough understanding of where and when specific inconsistencies arise. On the other hand, it also enables a quantitative assessment of the impact of introducing a specific inconsistency management strategy to the process.

Here, we focus on how inconsistencies should be managed in collaborative, heterogeneous settings in order to guarantee the correctness of the end product, yet enabling an efficient flow of engineering work. We present PROxIMA, a process-oriented framework for managing inconsistencies. The main features of the framework are (i) its support for the optimal selection of inconsistency management techniques across the process, and (ii) enforcing the overall consistency of the virtual product by process enactment and tool interoperability. The feasibility of our approach has been demonstrated by means of the engineering of an Autonomous Guided Vehicle (AGV), shown in Figure 1.

MANAGING INCONSISTENCIES

Inconsistencies may have different causes. Human error is the top cause, with examples such as making design mistakes and communication shortcomings. In more elaborate cases, inconsistencies may arise from imprecise semantics of modeling languages or from a mismatch between the semantics of different modeling languages (Huzar et al., 2005).

As proposed by Finkelstein (2000), instead of simply just removing inconsistencies from the design, we need to think about "managing consistency". One of the core activities of (in-)consistency management is the detection of inconsistencies. It is preferred that detection is achieved as early as possible, because the earlier the inconsistencies are detected, the lower the costs of resolving them. Early detection of inconsistencies also provides more freedom in choosing the appropriate resolution and tolerance techniques Van Der Straeten (2005); Dávid et al. (2016b).

A significant amount of research has been dedicated to solving semantic inconsistencies on the syntactic level (for example (Adourian and Vangheluwe, 2007; Bhave et al., 2010)). These approaches, however, are prone to lose vital information during the approximation and abstraction steps taken while translating semantic properties to linguistic structures and parameters. We argue that reasoning over explicitly modeled semantic properties suits the problem of tackling inconsistencies better, as demonstrated by, e.g., Herzig and Paredis (2014).

Prevention

Preventive approaches advocate avoiding inconsistencies before they occur. A pertinent example is using locking mechanisms in collaborative modeling settings, for example, by storing the models into a version control system. Such a technique, however, does not allow true parallelism in the engineering process, as entire models/files are locked for one stakeholder, putting others' work on hold, thus rendering the process less performant. Property-based locking (Chechik et al., 2015) alleviates this problem by reducing the scope of the lock to selected parts of the model, by the use of graph patterns. Additionally, such a fine-grained locking mechanism solves the problem of overlocking, where multiple stakeholders would lock too many models/files and a deadlock or livelock could occur. To support true parallelism in the engineering flow, contract-based techniques have been recently proposed (Vanherpen et al., 2016). In this approach, the accepted value ranges of the various system parameters are being negotiated by the stakeholders concerned with the respective parameters, and the negotiated values are persisted in a contract. The contract asserts what different stakeholders guarantee towards the others with respect to a specific parameter. By establishing a contract prior to the actual engineering work, the consistency of the virtual product, with respect to the properties detailed in the contract, can be guaranteed.

Allow-and-resolve

Instead of spending effort on the a priori negotiation of a contract, other approaches deal with inconsistencies in situ, i.e., inconsistencies are allowed to occur and subsequently are detected and later resolved, with the possibility of tolerating them (Balzer, 1991) for a certain period. After that period, the inconsistencies are either naturally resolved or need to be resolved explicitly. Dependency analysis between models has been widely applied in collaborative settings, where inconsistencies typically occur due to the usage of different overlapping stakeholder models (Qamar et al., 2012). In such techniques, a pivot model (correspondence model) relates the elements of the models and by this, enables propagating change information from one model to the others. In its most basic form, such a technique can provide a stakeholder with the information that a change in another stakeholder's model may have resulted in an inconsistency, and consequently, investigation/resolution of the problem is required. If more information about the relationship between the models (such as algebraic relationships between their parameters) is known, propagation of the change information may also trigger automated resolution of the inconsistency. SysML is a typical choice for establishing pivot models at the architectural level. Recently, the combination of SysML and ontological reasoning has been proposed in order to support the detection of semantic inconsistencies (Feldmann et al., 2014).

To manage an inconsistency, one has to choose an appropriate management technique, and to decide when to apply it. This decision can be viewed as an optimization problem and as such, requires an objective function. Typical examples include minimizing the time to market, minimizing the overall cost per work item, maximizing the utilization of resources, etc. Whichever metric we choose as an objective function for the optimization, the only way to derive them is by modeling and simulating the underlying engineering process. To satisfy the objective function, the process is transformed to an improved one, where every inconsistency is managed.

The challenge: appropriate selection of inconsistency management techniques

The management of inconsistencies is achieved by selecting an appropriate management pattern for each inconsistency instance. The selection is made from a catalogue of management patterns, such as reordering activities, introducing automated or manual consistency inspections after a critical region of the process, applying an engineering contract (Sangiovanni-Vincentelli et al., 2012) on a specific part of the process, etc.

Since the same type of inconsistency may be managed by applying different different management patterns, the selection of the most appropriate one should happen through quantified cost measures. We use the expected transit time of the process (to approximate the time-to-market) as a cost measure and prefer processes with lower transit time, as they result in accelerated product development. This problem is translated to a constraint solving and optimization problem which finds the best process alternative while managing every potential source of inconsistencies, discussed in the following section.

THE PROxIMA FRAMEWORK

The PROxIMA (PRocess Optimization + Inconsistency MAnagement) (Dávid et al., 2016a, 2017) framework provides tools for modeling numerous aspects of the engineering process (Figure 2); transforming the process into an optimized form; and enacting the optimized process while interfacing with engineering tools. The most important capabilities of the framework are

- modeling the engineering process with both (control) flow of engineering activities, and (data) flow of engineering artifacts in it;
- modeling the attributes and properties of the engineered system, and relating them to engineering activities and artifacts;

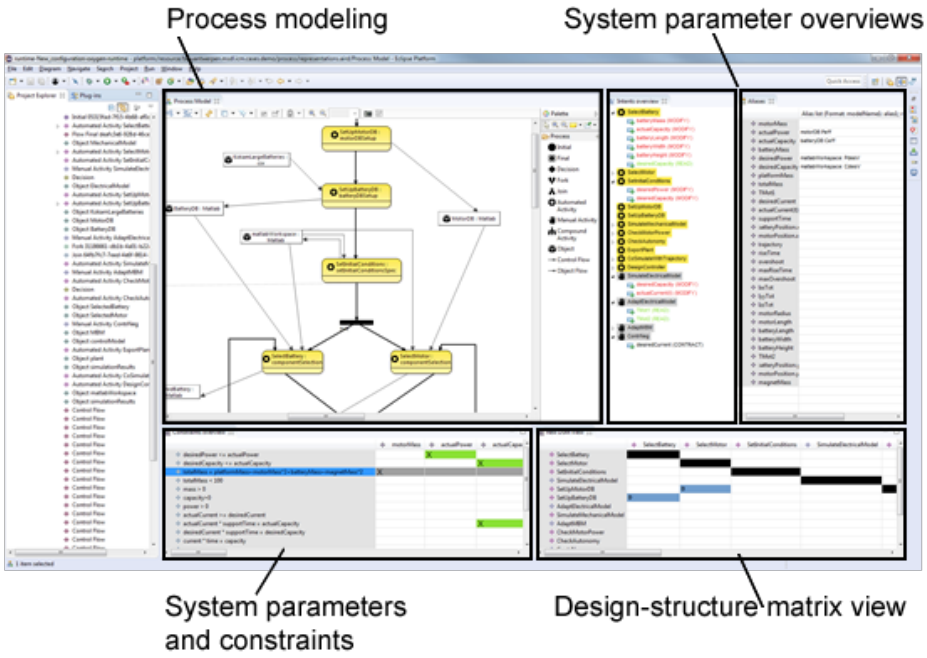

Figure 2: The modeling tool allows specifying various aspects of the underlying engineering process.

- resource and cost modeling;
- process optimization – finding the most cost-efficient process which still guarantees the correctness of the product by satisfying every system property;
- process enactment – exection and monitoring of the optimized process.

The tool was built on the Eclipse platform, using Python libraries for symbolic mathematical evaluations and tool interoperability.[1]
In the following, we discuss the typical steps required for inconsistency management in our framework.

Modeling the process

The modeling of the process is carried out using a visual modeling tool, built with the Sirius framework. Activities, their precedence, the input and output models are captured using the FTG+PM formalism (Lúcio et al., 2013). The formalism enables the usage of process models ("PM") in conjunction with the model of languages and transformations (the formalism-transformation graph - "FTG") used throughout the process. Formalisms and transformations serve as types to the processes: objects of the process are instances of languages of the FTG; and activities of the process realize transformations. In addition, system parameters which are subjects to potential inconsistencies, are charted as well and linked to the engineering activities which interact with them (e.g., by reading the value of the parameter or modifying it). This information is vital in identifying the root causes and scope of inconsistencies. In addition, resources (human and automated) can be modeled as well, such as stakeholders who enact the different activities, machines, tools, licenses, etc.
The optimality of the process is expressed by cost models. Multiple types of costs can be associated with activities, models, and tools. These costs are treated as different objectives to the optimization, resulting in a Pareto-front.

Off-line inconsistency management: process optimization by design-space exploration

After the process is modeled, the first phase of the inconsistency management takes place. The process is transformed so that as much inconsistencies are managed as possible, but also resulting in the best performing process, and respecting the resource constraints. The search for the resulting process is carried out using rule-based multi-objective design space exploration (DSE), in which the space of different process alternatives is

[1]The tool is available as open-source software (EPL) from `http://istvandavid.com/icm`.

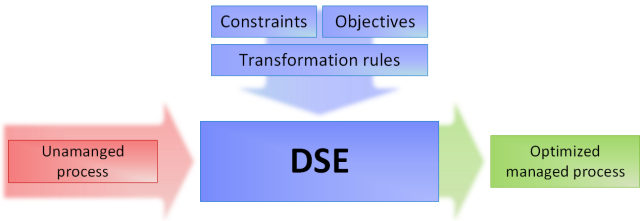

Figure 3: Design-Space Exploration.

searched to find the most appropriate one (Figure 3). The DSE engine takes the original, *unmanaged* and produces an *optimized managed* process as an output. This is achieved by applying pre-defined *model transformations rules* on the original process model in multiple iterations, thus constituting a search plan. Instead of an exhaustive approach, this search is governed by a hill climbing algorithm. There are two types of transformations:

- management transformations transform the process by applying management patterns onto it (e.g., introducing a contract negotiation activity in the process);
- optimization transformations, which try to parallelize the process as much as possible, also considering resource and scheduling constraints.

In every iteration, the consistency and performance of the new process is assessed so it can be decided whether the new process is an appropriate one. Managing inconsistencies is a hard *constraint* of the optimization, while maximizing the overall efficiency is a soft *objective*. This means, every potential inconsistency will be managed in the process, but this may sacrifice efficiency. Due to the stochastic nature of the process, assessing the performance of a candidate process is not straightforward. The process model is mapped onto the queueing network formalism of MathWorks' SimEvents and simulated to obtain a quantitative performance metric (Figure 4).
The following rules are used to map the engineering process onto the SimEvents formalism.

- **Activities** are translated to a Server processing a single token in the SimEvents formalism. The service time of the token in the server is based on the provided cost. The service time is either a constant value or a value sampled from a distribution.
- **Fork** nodes are translated to replicate nodes that process an incoming token (and all of it properties, like the total service time) to each of the outgoing branches.
- The **Join** node uses a combination of queues to let the tokens wait for the other branches. An entity combiner combines all tokens when they are available.
- **Decision** nodes are translated to an output switch element that routes to the available paths. The chosen path is sampled from the information provided in the process model. The merge node uses a path combiner to combine the incoming paths.

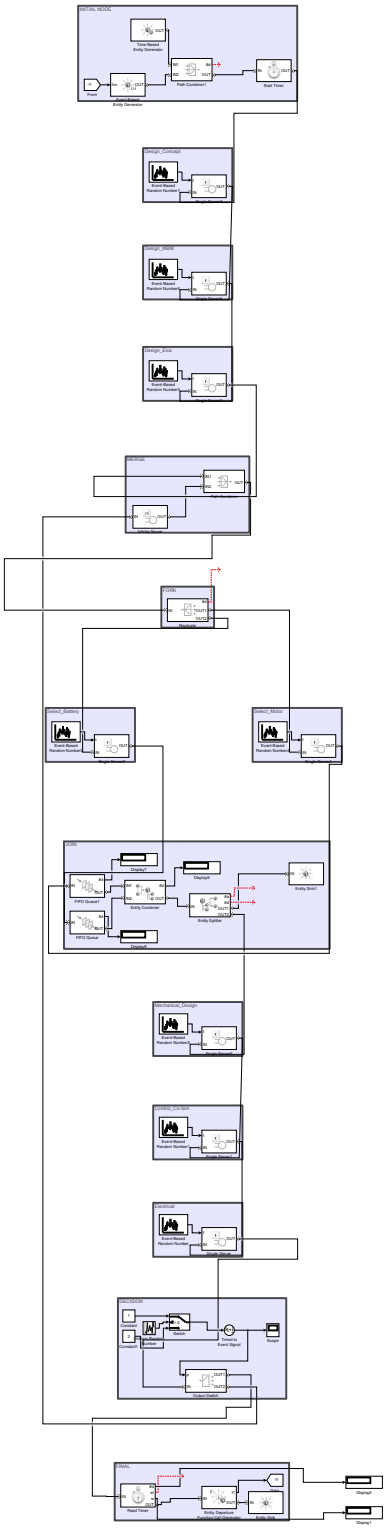

Figure 4: A quantitative SimEvents simulation model, generated from the original FTG+PM process model.

- Because we only allow for a **single process** to be executed at the same time, the logic in the final node allows for the creation of a new token in the initial node.
- The **control flow** between these newly created entities is equal to the control flow edges in the process model.

For each of the tokens, the total service time is recorded. As SimEvents is a stochastic formalism, multiple tokens are used to calculate the total cost of the process. We use the average service time of all tokens as the cost of the process.

On-line inconsistency management: process enactment and monitoring

Process enactment is commonly defined as the use of software to support the execution of operational processes, which enables mixing automated and manual activities. (CMMI Product Team, 2010) The PROxIMA framework provides an engine for enacting previously defined (and optimized) processes with the additional support for interoperability with a selection of engineering tools, such as Matlab/Simulink or Papyrus. During the enactment, the constraints of the system's parameters are continuously monitored. Whenever a violation of a constraint occurs, the stakeholders are notified about the occurrence of an inconsistency.
By employing symbolic mathematics, constraints can additionally be maintained in an incremental fashion and used for guiding the engineering work. Whenever the value of a system attribute can be derived from a combination of constraints and previously assigned attributes, the engine will provide this information to the stakeholder, thus aiding engineering decisions.

RELATED WORK

Inconsistency management has been a topic of interest for a long time in the domains of software engineering, mechatronic design and cyber-physical systems, due to their typically multi-view approach to system design. Di Ruscio et al. (2017) identify the research directions, challenges, and opportunities of collaborative MDSE and conclude that inconsistency management is one of the main enablers of efficient collaboration.
Multiple authors point out that managing inconsistencies should be carried out with processes in mind as well. Persson et al. (2013) identify consistency between the various views of cyber-physical system design as one of the main challenges in design of such complex systems. This is due to relations between views, with respect to their semantic relations, process and operations which often overlap. Multi-paradigm modeling Vangheluwe et al. (2002) advocates using the most appropriate formalism(s), at the most appropriate level(s) of abstraction, while also explicitly modelling the processes manipulating the models. The framework presented in this paper aims at the problem of semantics inconsistencies with a focus on processes.
Other approaches also acknowledge the role of semantic techniques in inconsistency management, and try to relate semantic concepts to the linguistic concepts of modeling. Hehenberger et al. (2010) organize structural design elements and their relations into a domain ontology to identify inconsistencies. A limited set of semantic properties are expressed with linguistic concepts which enables reasoning over semantic overlaps to a sufficient extent. Similarly, Chechik et al. (2015) introduce the notion of approximate properties: linguistic properties expressed as graph patterns which are accurate enough to appropriately approximate a semantic property. Approximate properties suitable to implement smart locking mechanisms in collaborative model-based design as they introduce a trade-off between the computational resources to obtain or check a property, and the accuracy of the results. As opposed to these, our approach makes semantic properties first-class artifacts and relates them to processes, instead of linguistic model elements, which enables management of a richer class of inconsistencies.
Ontologies have been used for inconsistency management by Kovalenko et al. (2014) to support automated detection of defects between domain-specific models. Similarly, Feldmann et al. (2014) use the OWL language in conjunction with a SysML-based approach to formally represent the design of a production system and evaluate the compatibility of domain-specific models in a collaborative setting. These approaches are complementary to ours: incorporating relationships between ontological properties for reasoning over inconsistencies is a planned extension to our work.
As opposed to the above techniques, inconsistency management in collaborative modeling is more frequently addressed on the linguistic level. Qamar et al. (2012) approach inconsistency management by making inter- and intra-model dependencies explicit. Dependencies are direct results of semantic overlaps and are used to notify stakeholders about possible inconsistencies when dependent properties change. Our approach introduces an indirection between models and properties by relating them to specific activities that during working over models also access properties with specific intents. Blanc et al. (2008) approach the detection of inconsistencies from a model operation based point of view, where models are stored as sequences of change events and inconsistencies are expressed in terms of Create Read Update Delete (CRUD) operations. Our approach generalizes this approach by introducing intents that are analogous with model operations, but they express change operations in terms of activities and properties.
In our work, we opted for the FTG+PM formalism for modeling processes. As compared to the widely used BPMN2.0 Object Management Group (OMG) (2011) or

BPEL-based process modeling frameworks (e.g., jBPM), our formalism allows modeling details more relevant to engineering scenarios in MDE settings. Models and transformations are first-class citizens in the FTG+PM, which enables deeper understanding of inconsistencies and more control over the enacted process.

Our framework provides simple undo/redo actions to revert to the latest consistent state of the models, but there have been other approaches to inconsistency resolution published. Mens et al. (2006) propose expressing the steps of inconsistency detection and resolution as graph transformation rules. Critical pair analysis is used to analyse potential dependencies between the detection and resolution of inconsistencies. It is, however, unclear whether critical pair analysis scales to industrial-size problems. Eramo et al. (2016) present an approach where each of the consistent alternatives are maintained throughout the process and removed when a decision is made and an alternative becomes infeasible. Almeida da Silva et al. (2010) investigate the possibilities of managing deviations of enacted processes from their respective specifications. It is not within the scope of our work, but indeed, deviations from the specified process are big threat to the validity of any process-oriented engineering approach. The efforts put into analyzing and optimizing a process model can be easily undone by deviating from (and sometimes even completely abandoning) the specification of the process.

CONCLUSIONS

In this paper, we presented an approach and framework for managing inconsistencies in collaborative and potentially highly parallelized, concurrent engineering settings. Our approach leverages the underlying engineering process and the various related information which enables reasoning over inconsistencies, their origins, impact and severity in a novel way.

We support the automated process of inconsistency management by a prototype tool. The approach has been evaluated over a case study of a mechatronic system, and Autonomous Guided Vehicle (AGV).

The approach discussed here can be an efficient enabler for collaborative engineering. At the same time, however, making the various facets of the engineering process explicit is a labor intensive and tedious task, which heavily frontloads the project. To alleviate the costs, the task of process modeling can be automated to a relatively high extent. The fundamental structure of the engineering process can be derived from business level processes.

In the future, we plan to augment our framework with performance simulation techniques for the process models. Our ongoing work focuses on performance metrics such as the transit time of the process. The impact of inconsistencies on the process is another performance metric we plan to support the simulation of. We also plan to enable explicit reasoning about the trade-off between managing inconsistencies in the process optimization phase (Dávid et al., 2016a) and during the enactment (Dávid et al., 2017). Another direction in our research is to support our approach with a library of inconsistency resolution techniques. We aim for developing a semi-automated selection of resolution methods, which will require detailed cost models of the process and all of its aspects.

ACKNOWLEDGEMENTS

This research has been partially funded by a BOF DOCPRO-1 grant of the University of Antwerp, Belgium.

REFERENCES

Adourian C. and Vangheluwe H., 2007. *Consistency Between Geometric and Dynamic Views of a Mechanical System.* In *Proc. of the 2007 Summer Computer Simulation Conf.* Society for Computer Simulation Int., San Diego, SCSC '07. ISBN 1-56555-316-0, 31:1–31:6.

Almeida da Silva M.A.; Bendraou R.; Blanc X.; and Gervais M.P., 2010. *Early Deviation Detection in Modeling Activities of MDE Processes*, Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-16129-2, 303–317. doi:10.1007/978-3-642-16129-2_22.

Balzer R., 1991. *Tolerating Inconsistency.* In *Proceedings of the 13th International Conference on Software Engineering.* IEEE Computer Society Press, ICSE '91. ISBN 0-89791-391-4, 158–165.

Bhave A.; Krogh B.; Garlan D.; and Schmerl B., 2010. *Multi-domain modeling of cyber-physical systems using architectural views. AVICPS 2010*, 43.

Blanc X.; Mounier I.; Mougenot A.; and Mens T., 2008. *Detecting model inconsistency through operation-based model construction.* In *Software Engineering, 2008. ICSE '08.* ISSN 0270-5257, 511–520. doi:10.1145/1368088.1368158.

Chechik M.; Dalpiaz F.; Debreceni C.; Horkoff J.; Ráth I.Z.; and Varró D., 2015. *Property-based methods for collaborative model development.*

CMMI Product Team, 2010. *CMMI for Development, Version 1.3, Tech. Rep. CMU/SEI-2010-TR-033.*

Corley J.; Syriani E.; Ergin H.; and Van Mierlo S., 2016. *Cloud-based multi-view modeling environments.* In *Modern Software Engineering Methodologies for Mobile and Cloud Environments*, IGI Global. 120–139.

Dávid I.; Denil J.; Gadeyne K.; and Vangheluwe H., 2016a. *Engineering Process Transformation to*

Manage (In)consistency. In *COMMitMDE 2016.* http://ceur-ws.org/Vol-1717/, 7–16.

Dávid I.; Meyers B.; Vanherpen K.; Van Tendeloo Y.; Berx K.; and Vangheluwe H., 2017. *Modeling and Enactment Support for Early Detection of Inconsistencies in Engineering Processes.* In *COMMitMDE 2017.*

Dávid I.; Syriani E.; Verbrugge C.; Buchs D.; Blouin D.; Cicchetti A.; and Vanherpen K., 2016b. *Towards Inconsistency Tolerance by Quantification of Semantic Inconsistencies.* In *COMMitMDE 2016.* 35–44.

Di Ruscio D.; Franzago M.; Muccini H.; and Malavolta I., 2017. *Envisioning the future of collaborative model-driven software engineering.* In *Proceedings of the 39th International Conference on Software Engineering Companion.* IEEE Press, 219–221.

Eramo R.; Pierantonio A.; and Rosa G., 2016. *Approaching Collaborative Modeling as an Uncertainty Reduction Process.* In *COMMitMDE 2016.* 27–34.

Feldmann S.; Kernschmidt K.; and Vogel-Heuser B., 2014. *Combining a SysML-based modeling approach and semantic technologies for analyzing change influences in manufacturing plant models. Procedia CIRP*, 17, 451–456.

Finkelstein A., 2000. *A Foolish Consistency: Technical Challenges in Consistency Management.* In *Database and Expert Systems Applications*, Springer, *LNCS*, vol. 1873. ISBN 978-3-540-67978-3, 1–5. doi: 10.1007/3-540-44469-6_1.

Hehenberger P.; Egyed A.; and Zeman K., 2010. *Consistency Checking of Mechatronic Design Models.* In *30th Computers and Information in Engineering Conf.* ASME, vol. 3. ISBN 978-0-7918-4411-3, 1141–1148. doi:10.1115/DETC2010-28615.

Herzig S.J. and Paredis C.J., 2014. *Bayesian Reasoning Over Models.* In *MoDeVVa 2014.* 69–78.

Herzig S.J.; Qamar A.; and Paredis C.J., 2014. *An approach to identifying inconsistencies in model-based systems engineering. Procedia Comp Sci*, 28, 354–362.

Huzar Z.; Kuzniarz L.; Reggio G.; and Sourrouille J.L., 2005. *Consistency Problems in UML-based Software Development.* In *Proceedings of the 2004 International Conference on UML Modeling Languages and Applications.* Springer-Verlag, Berlin, Heidelberg, UML'04. ISBN 3-540-25081-6, 1–12. doi:10.1007/978-3-540-31797-5_1.

Kovalenko O.; Serral E.; Sabou M.; Ekaputra F.J.; Winkler D.; and Biffl S., 2014. *Automating Cross- Disciplinary Defect Detection in Multi-Disciplinary Engineering Environments.* In *Knowledge Engineering and Knowledge Management*, Springer. 238–249.

Lúcio L.; Mustafiz S.; Denil J.; Vangheluwe H.; and Jukss M., 2013. *FTG+PM: An Integrated Framework for Investigating Model Transformation Chains.* In *SDL 2013: Model-Driven Dependability Engineering*, Springer, *LNCS*, vol. 7916. ISBN 978-3-642-38910-8, 182–202. doi:10.1007/978-3-642-38911-5_11.

Mens T.; Van Der Straeten R.; and D'Hondt M., 2006. *Detecting and resolving model inconsistencies using transformation dependency analysis.* In *International Conference on Model Driven Engineering Languages and Systems.* Springer Berlin Heidelberg, 200–214.

Object Management Group (OMG), 2011. *BPMN 2.0 Specification.* `http://www.bpmn.org/`. Accessed: 2018-05-06.

Persson M.; Törngren M.; Qamar A.; Westman J.; Biehl M.; Tripakis S.; Vangheluwe H.; and Denil J., 2013. *A Characterization of Integrated Multi-View Modeling in the Context of Embedded and Cyber-Physical Systems.* In *EMSOFT.* IEEE. ISBN 9781479914432, 1–10. doi:10.1109/EMSOFT.2013.6658588.

Qamar A.; Paredis C.J.; Wikander J.; and During C., 2012. *Dependency modeling and model management in mechatronic design. Journal of Computing and Information Science in Engineering*, 12, no. 4, 041009.

Sangiovanni-Vincentelli A.; Damm W.; and Passerone R., 2012. *Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems. European Journal of Control*, 18, no. 3, 217 – 238. ISSN 0947-3580. doi: http://dx.doi.org/10.3166/ejc.18.217-238.

Spanoudakis G. and Zisman A., 2001. *Inconsistency management in software engineering: Survey and open research issues.* In *in Handbook of Software Engineering and Knowledge Engineering.* World Scientific, 329–380.

Van Der Straeten R., 2005. *Inconsistency management in model-driven engineering. An Approach Using Description Logics (PhD thesis), Vrije Universiteit Brussel, Brussels, Belgium.*

Vangheluwe H.; De Lara J.; and Mosterman P.J., 2002. *An introduction to multi-paradigm modelling and simulation.* In *Proc. of the AIS'2002 conf.* 9–20.

Vanherpen K.; Denil J.; Dávid I.; Meulenaere P.D.; Mosterman P.J.; Torngren M.; Qamar A.; and Vangheluwe H., 2016. *Ontological reasoning for consistency in the design of cyber-physical systems.* In *CPPS 2016.* 1–8. doi:10.1109/CPPS.2016.7483922.